```
head   1.5;
access;
symbols;
locks; strict;
comment      @ * @;


1.5
date    97.07.28.21.50.41;    author jpalex; state Exp;
branches;
next    1.4;

1.4
date    97.07.28.21.06.23;    author jpalex; state Exp;
branches;
next    1.3;

1.3
date    97.07.18.21.32.19;    author jpalex; state Exp;
branches;
next    1.2;

1.2
date    97.07.15.18.42.21;    author jpalex; state Exp;
branches;
next    1.1;

1.1
date    97.07.15.18.32.50;    author jpalex; state Exp;
branches;
next    ;


desc
@@


1.5
log
@made it ogtstable
@
text
@#define NEW_FRAG_SPEC

#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <ieeefp.h>
```

```
#include <GL/glx.h>     /* this includes the necessary X and gl.h headers */
#include "GL/glfuture.h"
#include <GL/glu.h>
#ifndef ogtst
#include "xwin.h"
#include "xtrack.h"
#else
#include <ogtst.h>
#endif

#ifndef ogtst
#define testassert(a) if (!(a)) fprintf (stderr, \
    "assert %s failed in line %d of file %s\n", #a, __LINE__, __FILE__);
#else
#define testassert(a) if (!(a)) ogEnvLog (0, "assert %s failed in line %d of file %s\n", #a,
__LINE__, __FILE__);
#endif

#define min(a,b) ((a)<(b)) ? (a) : (b)

#ifdef file
#define makeimage readrgbaimage
#else
#define makeimage createimage
#endif

extern unsigned char  * readrgbaimage(char *iname, int *xsize, int *ysize);
extern GLfloat  * createimage(char *iname, int *xsize, int *ysize);
/* in float.c */
unsigned short FloatToS10E5B15(float c);
float S10E5B15ToFloat(unsigned short c);
float frameConvert (float original);
int isEqual (float f1, float f2);
void cleanUp(void);

#define toFrame FloatToS10E5B15
#define fromFrame S10E5B15ToFloat

void replace (float *f);

float maxval, minval;

GLfloat texcolor[4] = {4.5, 3.5, -2.5, 1.5};

#define NAN 3.5
#define INF 4.5
#define NEGINF -INF
#define NORM 2.22235
```

```
#define ZERO 0.0
#define MAX 5.5
#define MAXMINUSNORM 6.5
#define MIN -2.0
#define NUM_BLEND_TEST 17
#define NEGNORM -NORM

/* A, B */
GLfloat blend_input [NUM_BLEND_TEST][2] = {
 {INF, INF},
 {INF, NEGINF},
 {INF, ZERO},
 {INF, NORM},
 {NEGINF, INF},
 {NEGINF, NEGINF},
 {NEGINF, ZERO},
 {NEGINF, NORM},
 {ZERO, INF},
 {ZERO, NEGINF},
 {ZERO, ZERO},
 {ZERO, NORM},
 {NORM, INF},
 {NORM, NEGINF},
 {NORM, ZERO},
 {NAN, NORM},
 {NORM, NAN} };

/* parts of this table depend on NORM being positive */
/* A + B, A-B, A*B, 1/A */
/* can't test 1/A so we skip 'em */
float blend_output[NUM_BLEND_TEST][4] = {
 {MAX, ZERO, MAX, ZERO},
 {ZERO, MAX, MIN, ZERO},
 {MAX, MAX, ZERO, ZERO},
 {MAX, MAXMINUSNORM, MAX, ZERO},
 {ZERO, MIN, MIN, ZERO},
 {MIN, ZERO, MAX, ZERO},
 {MIN, MIN, ZERO, ZERO},
 {MIN, MIN, MIN, ZERO},
 {MAX, MIN, ZERO, NAN},
 {MIN, MAX, ZERO, NAN},
 {ZERO, ZERO, ZERO, NAN},
 {NORM, NEGNORM, ZERO, NAN},
 {MAX, MIN, MAX, 1.0/NORM},
 {MIN, MAX, MIN, 1.0/NORM},
 {NORM, NORM, ZERO, 1.0/NORM},
 {NAN, NAN, NAN, NAN},
 {NAN, NAN, NAN, NAN}
};
```

```
#define BOX_WIDTH 50
#define BOX_HEIGHT 50
void redrawColorRange (void)
{
#ifdef GL_SGIX_color_range
  GLfloat pix[4];
  GLfloat nolight [4] = {2.434242, -4.25636, 5.22424, 3.33442};
  GLfloat fnolight [4];
  GLfloat blend[4] = {-20.24233, 40.24244, .5353, -2.3545};
  GLfloat fblend[4];
  GLfloat nolight3 [4] = {3.24242, -2.394, 2.34424};
  GLfloat texfragcolor [4] = {5.9238, 2.2436, -.5, 2.11};
  GLfloat smag[4] = {2.01131, 2.2434, -2.034243, 2.0334909}; /* super-magenta */
  /*  GLint smagint[4] = {INT_MAX, INT_MAX, INT_MIN, INT_MAX};  */
  GLfloat bla[4] = {0., 0., 0., 1.};
  GLfloat opaquebla[4] = {0., 0., 0., -5.34252}; /* really opaque black */
  GLfloat envcolor[4] = {-4.11144, 4.242225, 10.384, 2.33330};
  GLfloat bordercolor[4] = {40.09083, -2.2434, 1.5242, 3.0333};
  GLfloat Aarray[4], Barray[4];
  GLfloat pixafter[4];
  /*  GLint   ipix[4];*/
  GLfloat A, B;
  int x,y, i;
  GLfloat whi[4] = {1., 1.,1., 1.};
  int xsize, ysize;
  GLfloat *image;
  GLint vsize[4];

  x = y = 0;

  glGetFloatv (GL_MIN_RED_SGIX, &minval);
  glGetFloatv (GL_MAX_RED_SGIX, &maxval);

  glGetIntegerv (GL_VIEWPORT, vsize);

  /* CHECK CLEARCOLOR */
  glClearColor(nolight[0], nolight[1], nolight[2], nolight[3]);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glReadPixels (0, 0, 1, 1, GL_RGBA, GL_FLOAT, pix);
  testassert ( (pix[0] == frameConvert(nolight[0])) &&
               (pix[1] == frameConvert(nolight[1])) &&
               (pix[2] == frameConvert(nolight[2])) &&
               (pix[3] == frameConvert(nolight[3])) );

  glClearColor(0., 0., 0., 0.);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
```

```
/* CHECK NO LIGHTING */
glColor4fv (nolight);
glRectf (-1., -1., 1., 1.);
glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);
testassert ( (pix[0] == frameConvert(nolight[0])) &&
             (pix[1] == frameConvert(nolight[1])) &&
             (pix[2] == frameConvert(nolight[2])) &&
             (pix[3] == frameConvert(nolight[3])) );

/* ...make sure default A is 1.0 */
x += 50;
glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
glColor3fv (nolight3);
glRectf (-1., -1., 1., 1.);
glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);
testassert ( (pix[0] == frameConvert(nolight3[0])) &&
             (pix[1] == frameConvert(nolight3[1])) &&
             (pix[2] == frameConvert(nolight3[2])) &&
             (pix[3] == frameConvert(1.0)) );

/* CHECK TEXTURES */
x += 50;
glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
image = makeimage("red1.rgb", &xsize, &ysize);
glColor4fv (texfragcolor);
glTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA, xsize, ysize,
              0, GL_RGBA, GL_FLOAT, image);
glEnable (GL_TEXTURE_2D);
glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glRectf (-1., -1., 1., 1.);
glReadPixels (x+5, y + BOX_HEIGHT/2, 1, 1, GL_RGBA, GL_FLOAT, pix);
testassert ( (pix[0] == frameConvert(texfragcolor[0] * texcolor[0])) &&
             (pix[1] == frameConvert(texfragcolor[1] * texcolor[1])) &&
             (pix[2] == frameConvert(texfragcolor[2] * texcolor[2])) &&
             (pix[3] == frameConvert(texfragcolor[3] * texcolor[3])) );

/* check env color clamping*/
x += 50;
glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_BLEND);
glTexEnvfv (GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, envcolor);
glRectf (-1., -1., 1., 1.);
glReadPixels (x+1, y + BOX_HEIGHT/2, 1, 1, GL_RGBA, GL_FLOAT, pix);
testassert ( (pix[0] == frameConvert(texfragcolor[0] * (1.0-texcolor[0]) + envcolor[0] *
texcolor[0])) &&
             (pix[1] == frameConvert(texfragcolor[1] * (1.0-texcolor[1]) + envcolor[1] *
```

```
texcolor[1])) &&
            (pix[2] == frameConvert(texfragcolor[2] * (1.0-texcolor[2]) + envcolor[2] *
texcolor[2])) &&
            (pix[3] == frameConvert(texfragcolor[3] * texcolor[3]) );

 /* check border color clamping */
 glTexParameterfv( GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, bordercolor);
 glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP); /* this makes
border show up*/
 glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
 x += 50;
 glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
 glRectf (-1., -1., 1., 1.);
 /* tex coords are 0 everywhere - I think - so it doesn't matter where we read, it's
  all edge. */
 /* LINEAR interpolation will give us half of the texture and half of the border; thus,
  the average. Because s is clamped but t is not, reading at the corner gives us
  (-1, 0), (-1, 63), (0, 0), (0, 63) because the t's wrap and the s's don't. Thus
  it's half border & half texture*/
 glReadPixels (x+1, y + BOX_HEIGHT/2, 1, 1, GL_RGBA, GL_FLOAT, pix);
 testassert ( (pix[0] == frameConvert((texcolor[0] + bordercolor[0]) * 0.5)) &&
            (pix[1] == frameConvert((texcolor[1] + bordercolor[1]) * 0.5)) &&
            (pix[2] == frameConvert((texcolor[2] + bordercolor[2]) * 0.5)) &&
            (pix[3] == frameConvert((texcolor[3] + bordercolor[3]) * 0.5)) );

 glDisable (GL_TEXTURE_2D);


 glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, bla);
 glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, smag); /* need to use diffuse
because the alpha value of the final color is taken from diffuse alpha and we want to check
clamping of all r, g, b, a*/
 glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, smag);
 glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 30.);

 /* CHECK PER-VERTEX LIGHTING */
 x += 50;
 glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
 glEnable (GL_LIGHTING);
 glEnable (GL_LIGHT0);

 glLightfv(GL_LIGHT0, GL_AMBIENT, bla);
 glLightfv(GL_LIGHT0, GL_DIFFUSE, smag);
 glLightfv(GL_LIGHT0, GL_SPECULAR, smag);

 glRectf (-1., -1., 1., 1.);
 glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);

 /*... spec mat = spec light= diffuse light = diffuse mat == smag
```

```
    so spec mat * spec light + diffuse mat * diffuse light == smag * smag + smag * smag
    which is what we check against */
  testassert ( (pix[0] == frameConvert(smag[0] * smag[0] * 2.)) &&
               (pix[1] == frameConvert(smag[1] * smag[1] * 2.)) &&
               (pix[2] == frameConvert(smag[2] * smag[2] * 2.)) &&
               (pix[3] == frameConvert(smag[3])) ); /* alpha doesn't get multiplied */

  /* CHECK GET */
  glGetMaterialfv (GL_FRONT, GL_DIFFUSE, pix);
  testassert ( (pix[0] == smag[0]) &&
               (pix[1] == smag[1]) &&
               (pix[2] == smag[2]) &&
               (pix[3] == smag[3]) );


  /* CHECK FRAG LIGHTING */
#ifdef GL_SGIX_fragment_lighting
  glEnable (GL_FRAGMENT_LIGHTING_SGIX);
  glEnable (GL_FRAGMENT_LIGHT0_SGIX);
  glLightEnviSGIX(GL_LIGHT_ENV_MODE_SGIX, GL_REPLACE);
  /* ...check the pass-through of per-vertex result into frag ambient */
  glFragmentColorMaterialSGIX (GL_FRONT_AND_BACK, GL_AMBIENT);
  glEnable (GL_FRAGMENT_COLOR_MATERIAL_SGIX);

  glFragmentMaterialfvSGIX(GL_FRONT_AND_BACK, GL_AMBIENT, bla);
  glFragmentMaterialfvSGIX(GL_FRONT_AND_BACK, GL_DIFFUSE, opaquebla);
  glFragmentMaterialfvSGIX(GL_FRONT_AND_BACK, GL_SPECULAR, bla);

  /* ..scene ambient was irrelevant in per-vertex tests because the ambient
     material was black; it'll get turned on in this case so we
     explicitly turn it off here. */
#ifdef NEW_FRAG_SPEC
  glFragmentLightModelfvSGIX (GL_FRAGMENT_LIGHT_MODEL_AMBIENT_SGIX, bla);
#endif
  glLightModelfv (GL_LIGHT_MODEL_AMBIENT, bla);

  glFragmentMaterialiSGIX(GL_FRONT_AND_BACK, GL_ENV_MAP_SGIX, GL_NONE);

  glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_AMBIENT, whi);
  glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_DIFFUSE, whi);
  glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_SPECULAR, whi);

  x += 50;
  glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
  glRectf (-1., -1., 1., 1.);
  glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);
  /* ...should be same as per-vertex result (since we're just passing it thru)
     except that alpha matches the per-fragment diffuse alpha (from opaquebla)
     instead of the per-vertex alpha */
```

```
 testassert ( (pix[0] == frameConvert(smag[0] * smag[0] * 2.)) &&
              (pix[1] == frameConvert(smag[1] * smag[1] * 2.)) &&
              (pix[2] == frameConvert(smag[2] * smag[2] * 2.)) &&
              (pix[3] == frameConvert(opaquebla[3])) ); /* alpha doesn't get multiplied */

 /* ...check general fragment lighting */
 glDisable (GL_FRAGMENT_COLOR_MATERIAL_SGIX);
 glDisable (GL_LIGHTING);
 glDisable (GL_LIGHT0);

 glFragmentMaterialfvSGIX(GL_FRONT_AND_BACK, GL_AMBIENT, bla);
 glFragmentMaterialfvSGIX(GL_FRONT_AND_BACK, GL_DIFFUSE, smag); /* need to use
diffuse because the alpha value of the final color is taken from diffuse alpha and we want to
check clamping of all r, g, b, a*/
 glFragmentMaterialfvSGIX(GL_FRONT_AND_BACK, GL_SPECULAR, smag);
 glFragmentMaterialfSGIX(GL_FRONT_AND_BACK, GL_SHININESS, 30.);

 glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_AMBIENT, bla);
 glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_DIFFUSE, smag);
 glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_SPECULAR, smag);

 x += 50;
 glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
 glRectf (-1., -1., 1., 1.);
 glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);

 /* ...spec mat = spec light= diffuse light = diffuse mat == smag
   so spec mat * spec light + diffuse mat * diffuse light == smag * smag + smag * smag
   which is what we check against */
 testassert ( (pix[0] == frameConvert(smag[0] * smag[0]) * 2.) &&
              (pix[1] == frameConvert(smag[1] * smag[1]) * 2.) &&
              (pix[2] == frameConvert(smag[2] * smag[2]) * 2.) &&
              (pix[3] == frameConvert(smag[3])) ); /* alpha doesn't get multiplied */

 /* ...test cube map / environment term */
 glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_DIFFUSE, bla);
 glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_SPECULAR, bla);
 /* ...image here was created up above */
 gluBuild2DMipmaps(GL_CUBE_MAP_ZP_SGIX, GL_RGBA, xsize, ysize, GL_RGBA,
                   GL_FLOAT, image);
 gluBuild2DMipmaps(GL_CUBE_MAP_ZN_SGIX, GL_RGBA, xsize, ysize, GL_RGBA,
                   GL_FLOAT, image);
 gluBuild2DMipmaps(GL_CUBE_MAP_YP_SGIX, GL_RGBA, xsize, ysize, GL_RGBA,
                   GL_FLOAT, image);
 gluBuild2DMipmaps(GL_CUBE_MAP_YN_SGIX, GL_RGBA, xsize, ysize, GL_RGBA,
                   GL_FLOAT, image);
 gluBuild2DMipmaps(GL_CUBE_MAP_XP_SGIX, GL_RGBA, xsize, ysize, GL_RGBA,
                   GL_FLOAT, image);
 gluBuild2DMipmaps(GL_CUBE_MAP_XN_SGIX, GL_RGBA, xsize, ysize, GL_RGBA,
```

```
                    GL_FLOAT, image);
  glEnable (GL_CUBE_MAP_SGIX);
  glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
  glFragmentMaterialiSGIX(GL_FRONT_AND_BACK, GL_ENV_MAP_SGIX,
GL_CUBE_MAP_SGIX);

  x += 50;
  glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
  glRectf (-1., -1., 1., 1.);
  glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);
  /* ...all the lights are off. The only term on is spec mat * env color */
  testassert ( (pix[0] == frameConvert(smag[0] * texcolor[0])) &&
               (pix[1] == frameConvert(smag[1] * texcolor[1])) &&
               (pix[2] == frameConvert(smag[2] * texcolor[2])) &&
               (pix[3] == frameConvert(smag[3])) );
  glDisable (GL_CUBE_MAP_SGIX);

  /* CHECK GETS */
  glGetFragmentMaterialfvSGIX (GL_FRONT, GL_DIFFUSE, pix);
  testassert ( (pix[0] == smag[0]) &&
               (pix[1] == smag[1]) &&
               (pix[2] == smag[2]) &&
               (pix[3] == smag[3]) );


  glDisable (GL_FRAGMENT_LIGHTING_SGIX);
  glDisable (GL_FRAGMENT_LIGHT0_SGIX);
#endif

  glDisable (GL_LIGHTING);
  glDisable (GL_LIGHT0);

  /* CHECK BLENDING */
  y+=BOX_HEIGHT; x = 0;
  glEnable (GL_BLEND);

  /* ...adding */
  glBlendFunc (GL_ONE, GL_ONE);
  glBlendEquationEXT (GL_FUNC_ADD_EXT);
  glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
  glColor4fv (nolight);
  glRectf (-1., -1., 1., 1.);
  glColor4fv (blend);
  glRectf (-1., -1., 1., 1.); /* should add on */
  glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);
  for (i=0; i <4; i++)
    {
    fnolight[i] = frameConvert (nolight[i]);
    fblend[i] = frameConvert (blend[i]);
```

```
  }

  /* so nolight is written in and then read out (one convert) to be added to blend and then
written in.  (and then read out by ReadPixels, the second convert)*/
  testassert ( (pix[0] == frameConvert(fnolight[0] + blend[0])) &&
               (pix[1] == frameConvert(fnolight[1] + blend[1])) &&
               (pix[2] == frameConvert(fnolight[2] + blend[2])) &&
               (pix[3] == frameConvert(fnolight[3] + blend[3])) );

  /* one-minus action */
  x+=BOX_WIDTH;
  glDisable (GL_BLEND);
  glBlendFunc (GL_ONE_MINUS_DST_COLOR, GL_ONE_MINUS_SRC_COLOR);
  glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
  glColor4fv (nolight);
  glRectf (-1., -1., 1., 1.);
  glEnable (GL_BLEND);
  glColor4fv (blend);
  glRectf (-1., -1., 1., 1.); /* should add on */
  glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);
  testassert ( (pix[0] == frameConvert(fnolight[0] * (1.0 - blend[0]) + blend[0] * (1.0 -
fnolight[0]))) &&
               (pix[1] == frameConvert(fnolight[1] * (1.0 - blend[1]) + blend[1] * (1.0 -
fnolight[1]))) &&
               (pix[2] == frameConvert(fnolight[2] * (1.0 - blend[2]) + blend[2] * (1.0 -
fnolight[2]))) &&
               (pix[3] == frameConvert(fnolight[3] * (1.0 - blend[3]) + blend[3] * (1.0 -
fnolight[3]))) );

  /* ...subtract */
  x+=BOX_WIDTH;
  glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
  glDisable (GL_BLEND);
  glColor4fv (blend);
  glRectf (-1., -1., 1., 1.);
  glBlendFunc (GL_ONE, GL_ONE);
  glBlendEquationEXT (GL_FUNC_SUBTRACT_EXT);
  glEnable (GL_BLEND);
  glColor4fv (nolight);
  glRectf (-1., -1., 1., 1.); /* should subtract */
  glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);
  testassert ( (frameConvert (nolight[0] - fblend[0]) == pix[0]) &&
               (frameConvert (nolight[1] - fblend[1]) == pix[1]) &&
               (frameConvert (nolight[2] - fblend[2]) == pix[2]) &&
               (frameConvert (nolight[3] - fblend[3]) == pix[3]) );

  /* min */
  x+=BOX_WIDTH;
  glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
```

```
  glDisable (GL_BLEND);
  glColor4fv (blend);
  glRectf (-1., -1., 1., 1.);
  glBlendFunc (GL_ONE, GL_ONE);
  glBlendEquationEXT (GL_MIN_EXT);
  glEnable (GL_BLEND);
  glColor4fv (nolight);
  glRectf (-1., -1., 1., 1.); /* should subtract */
  glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);
  testassert ( (min (nolight[0], fblend[0]) == pix[0]) &&
               (min (nolight[1], fblend[1]) == pix[1]) &&
               (min (nolight[2], fblend[2]) == pix[2]) &&
               (min (nolight[3], fblend[3]) == pix[3]) );

#define test4error(a,b,c,d, e) fprintf (stderr, a, b, c, d, e);

  x+=BOX_WIDTH;

  /* FLOATING-POINT SPEC - Inf, NaN, 0 interactions*/
  for (i = 0; i < NUM_BLEND_TEST; i++)
   {
     replace (&blend_input[i][0]);
     replace (&blend_input[i][1]);
     replace (&blend_output[i][0]);
     replace (&blend_output[i][1]);
     replace (&blend_output[i][2]);
     replace (&blend_output[i][3]);

   }

  for (i = 0; i < NUM_BLEND_TEST; i++)
   {
     A = blend_input[i][0];
     B = blend_input[i][1];

     Aarray[0] = Aarray[1] = Aarray[2] = Aarray[3] = A;
     Barray[0] = Barray[1] = Barray[2] = Barray[3] = B;
     glViewport (x, y+i, 1, 1);
     glDisable (GL_BLEND);
     glColor4fv (Aarray);
     glRectf (-1., -1., 1., 1.);
     glBlendFunc (GL_ONE, GL_ONE);
     glBlendEquationEXT (GL_FUNC_ADD_EXT);
     glEnable (GL_BLEND);
     glColor4fv (Barray);
     glRectf (-1., -1., 1., 1.);
     glReadPixels (x, y+i, 1, 1, GL_RGBA, GL_FLOAT, pix);

     /* need to use special function to compare NaN's */
```

```
    if ( !isEqual (pix[0], blend_output[i][0]))
        {
          test4error ("%f + %f gave result %f and not the proper %f\n", A, B, pix[0],
blend_output[i][0]);
        }

    glViewport (x+1, y + i, 1, 1);
    glDisable (GL_BLEND);
    glColor4fv (Aarray);
    glRectf (-1., -1., 1., 1.);
    glBlendFunc (GL_ONE, GL_ONE);
    glBlendEquationEXT (GL_FUNC_REVERSE_SUBTRACT_EXT);
    glEnable (GL_BLEND);
    glColor4fv (Barray);
    glRectf (-1., -1., 1., 1.);
    glReadPixels (x+1, y+i, 1, 1, GL_RGBA, GL_FLOAT, pix);

    if ( !isEqual (pix[0], blend_output[i][1]))
        {
          test4error ("%f - %f gave result %f and not the proper %f\n", A, B, pix[0],
blend_output[i][1]);
        }

    glViewport (x+2, y+i, 1, 1);
    glDisable (GL_BLEND);
    glColor4fv (Aarray);
    glRectf (-1., -1., 1., 1.);
    glBlendFunc (GL_DST_COLOR, GL_ZERO); /* does the mult */
    glBlendEquationEXT (GL_FUNC_ADD_EXT); /* don't be fooled; we are multiplying*/
    glEnable (GL_BLEND);
    glColor4fv (Barray);
    glRectf (-1., -1., 1., 1.);
    glReadPixels (x+2, y+i, 1, 1, GL_RGBA, GL_FLOAT, pix);

    if ( !isEqual (pix[0], blend_output[i][2]))
        {
          test4error ("%f * %f gave result %f and not the proper %f\n", A, B, pix[0],
blend_output[i][2]);
        }

  }

 /* glBlendColorEXT */
 glBlendColorEXT (blend[0], blend[1], blend[2], blend[3]);
 glBlendFunc (GL_CONSTANT_COLOR_EXT, GL_ZERO);
 glBlendEquationEXT (GL_FUNC_ADD_EXT);
 x+=BOX_WIDTH;
 glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);
 glColor4fv (nolight);
```

```
  glRectf (-1., -1., 1., 1.);
  glReadPixels (x+1, y + BOX_HEIGHT / 2, 1, 1, GL_RGBA, GL_FLOAT, pix);
  testassert ( (pix[0] == frameConvert(blend[0] * nolight[0])) &&
               (pix[1] == frameConvert(blend[1] * nolight[1])) &&
               (pix[2] == frameConvert(blend[2] * nolight[2])) &&
               (pix[3] == frameConvert(blend[3] * nolight[3])) );

  glDisable (GL_BLEND);

  x+= BOX_WIDTH;
  glViewport (x, y, BOX_WIDTH, BOX_HEIGHT);

  /* CHECK DRAWPIXEL/COPYPIXEL */
  /* not drawing where we expect */
  glRasterPos3i (0, 0, 0); /* squarely in the middle of the viewport */
  glDrawPixels (1, 1, GL_RGBA, GL_FLOAT, blend);
  glReadPixels (x + BOX_WIDTH/2, y+BOX_HEIGHT/2, 1, 1, GL_RGBA, GL_FLOAT,
pixafter);
  testassert ( (frameConvert(blend[0]) == pixafter[0]) &&
               (frameConvert(blend[1]) == pixafter[1]) &&
               (frameConvert(blend[2]) == pixafter[2]) &&
               (frameConvert(blend[3]) == pixafter[3]) );


/* ...copy from x+1 to current raster pos (== lower left) */
  glRasterPos3i (-1, -1, 0);
  glCopyPixels (x+BOX_WIDTH/2, y+BOX_HEIGHT/2, 1, 1, GL_COLOR);
  glReadPixels (x, y, 1, 1, GL_RGBA, GL_FLOAT, pixafter);
  testassert ( (frameConvert(blend[0]) == pixafter[0]) &&
               (frameConvert(blend[1]) == pixafter[1]) &&
               (frameConvert(blend[2]) == pixafter[2]) &&
               (frameConvert(blend[3]) == pixafter[3]) );


  glLightModelfv (GL_LIGHT_MODEL_AMBIENT, smag);
  glGetFloatv (GL_LIGHT_MODEL_AMBIENT, pix);
  testassert ( (smag[0] == pix[0]) &&
               (smag[1] == pix[1]) &&
               (smag[2] == pix[2]) &&
               (smag[3] == pix[3]) );

  /* float->int is not being handled correctly */
  /*  glGetIntegerv (GL_LIGHT_MODEL_AMBIENT, ipix);
  fprintf (stderr, "ipix: %d, %d, %d, %d\n", ipix[0], ipix[1], ipix[2], ipix[3]);
  testassert ( (smagint[0] == ipix[0]) &&
               (smagint[1] == ipix[1]) &&
               (smagint[2] == ipix[2]) &&
               (smagint[3] == ipix[3]) );
               */
```

```
#ifdef GL_SGIX_fragment_lighting
  glDisable (GL_FRAGMENT_LIGHTING_SGIX);
  glDisable (GL_FRAGMENT_LIGHT0_SGIX);
#endif
  glDisable (GL_LIGHTING);
  glDisable (GL_LIGHT0);

  glViewport (vsize[0], vsize[1], vsize[2], vsize[3]);
#endif
}


#ifdef ogtst
/*ARGSUSED*/
TESTMOD(crange)
{
#ifdef GL_SGIX_color_range
 redrawColorRange();
#endif
}
#else
/*ARGSUSED*/
void callback(long ks, short data)
{
 switch (ks)
   {
   case XREDRAW:
    break;
   default:
    return;
   }

 redrawColorRange ();
 cleanUp();
}


void main(int argc, char **argv)
{
 xprefsize(500, 320);
 xkeepaspect(1, 1);
 xwinopen("Extended Range & Precision Test");

 /* Now wait for the user to do something */
 for( ;; ) {
  xpolldevices(callback);

 }
```

```
}
#endif

#define WIDTH 64
#define HEIGHT 64
#define DEPTH 4

GLfloat daimage [WIDTH * HEIGHT * DEPTH];

/*ARGSUSED*/
GLfloat  * createimage(char *iname, int *xsize, int *ysize)
{
int x, y, offset;

offset = 0;
for (x = 0; x < WIDTH; x++)
 {
   for (y = 0; y < HEIGHT; y++)
     {
         daimage [offset+0] = texcolor[0];
         daimage [offset+1] = texcolor[1];
         daimage [offset+2] = texcolor[2];
         daimage [offset+3] = texcolor[3];
         offset += DEPTH;
     }
 }
*xsize = WIDTH;
*ysize = HEIGHT;
return daimage;
}

/* simulates being written to & read back from the framebuffer */
float frameConvert (float original)
{
return S10E5B15ToFloat(FloatToS10E5B15(original));
}

void replace (float *f)
{
     if (*f == INF)
          *f = 1./0.;
     else if (*f == NEGINF)
          *f = -1./0.;
     else if (*f == NAN)
          *f = 0./0.;
     else if (*f == MAX)
          *f = maxval;
     else if (*f == MAXMINUSNORM)
```

```
        *f = maxval - NORM;
    else if (*f == MIN)
        *f = minval;
    else
        *f = frameConvert (*f);
}

int isEqual (float f1, float f2)
{
  unsigned short s1, s2;
  if (f1 == f2) return 1;
  s1 = toFrame (f1); s2 = toFrame (f2);
  if (s1 == s2) return 1; /* this'll handle NaN comparisons */
  return 0;
}

#ifdef ogtst
CLEANUP(crange)
#else
void cleanUp(void)
#endif
{
#ifdef GL_SGIX_color_range
  GLfloat def_amb[4] = {.2, .2, .2, 1.};
  GLfloat def_dif[4] = {.8, .8, .8, 1.};
  GLfloat lightpos[4] = {.0, .0, 1., .0};
  GLfloat bla[] = {0., .0, .0, 1.0};
  GLfloat realbla[] = {0., .0, .0, 0.0};
  GLfloat whi[] = {1.0, 1.0, 1.0, 1.0};

  /* go back to defaults */
#ifdef GL_SGIX_fragment_lighting
    glFragmentLightfvSGIX (GL_FRAGMENT_LIGHT0_SGIX, GL_AMBIENT, bla);
    glFragmentLightfvSGIX (GL_FRAGMENT_LIGHT0_SGIX, GL_DIFFUSE, whi);
    glFragmentLightfvSGIX (GL_FRAGMENT_LIGHT0_SGIX, GL_SPECULAR, whi);
    glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_POSITION, lightpos);

    glFragmentLightfSGIX (GL_FRAGMENT_LIGHT0_SGIX, GL_SPOT_EXPONENT, 0.0);
    glFragmentLightfSGIX (GL_FRAGMENT_LIGHT0_SGIX, GL_SPOT_CUTOFF, 180.0);
    glFragmentLightfSGIX (GL_FRAGMENT_LIGHT0_SGIX,
GL_CONSTANT_ATTENUATION, 1.0);
    glFragmentLightfSGIX (GL_FRAGMENT_LIGHT0_SGIX, GL_LINEAR_ATTENUATION,
0.0);
    glFragmentLightfSGIX (GL_FRAGMENT_LIGHT0_SGIX,
GL_QUADRATIC_ATTENUATION, 0.0);
    glFragmentLightfSGIX (GL_FRAGMENT_LIGHT0_SGIX,
GL_SPOT_CUTOFF_DELTA_SGIX, 0.0);

    glFragmentMaterialfSGIX(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
```

```
    glFragmentMaterialfvSGIX (GL_FRONT_AND_BACK, GL_SPECULAR, bla);
    glFragmentMaterialfvSGIX (GL_FRONT_AND_BACK, GL_EMISSION, bla);
    glFragmentMaterialfvSGIX (GL_FRONT_AND_BACK, GL_AMBIENT, def_amb);
    glFragmentMaterialfvSGIX (GL_FRONT_AND_BACK, GL_DIFFUSE, def_dif);
#endif
    glLightfv (GL_LIGHT0, GL_AMBIENT, bla);
    glLightfv (GL_LIGHT0, GL_DIFFUSE, whi);
    glLightfv (GL_LIGHT0, GL_SPECULAR, whi);
    glLightfv(GL_LIGHT0, GL_POSITION, lightpos);

    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0);
    glMaterialfv (GL_FRONT_AND_BACK, GL_SPECULAR, bla);
    glMaterialfv (GL_FRONT_AND_BACK, GL_EMISSION, bla);
    glMaterialfv (GL_FRONT_AND_BACK, GL_AMBIENT, def_amb);
    glMaterialfv (GL_FRONT_AND_BACK, GL_DIFFUSE, def_dif);

    glTexEnvfv (GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, realbla);
    glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST_MIPMAP_LINEAR);
    glTexParameterf( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameterfv( GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, realbla);
    glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT); /* this
makes border show up*/
    glBlendColorEXT (bla[0], bla[1], bla[2], 0);
    glBlendEquationEXT (GL_FUNC_ADD_EXT);
    glTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA, 0, 0,
                  0, GL_RGBA, GL_FLOAT, NULL);

    glDisable (GL_FRAGMENT_COLOR_MATERIAL_SGIX);

    glDisable(GL_FRAGMENT_LIGHTING_SGIX);
    glDisable(GL_FRAGMENT_LIGHT0_SGIX);
    glDisable (GL_LIGHTING);
    glDisable (GL_LIGHT0);
    glColor4f(1., 1., 1., 1.);
    glNormal3f(.0, 0., 1.);
    glDisable (GL_DEPTH_TEST);
    glFrontFace (GL_CCW);

    glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, 0);
    glLightModelf(GL_LIGHT_MODEL_TWO_SIDE, 0);
#ifdef GL_SGIX_fragment_lighting
#ifdef NEW_FRAG_SPEC
    glFragmentLightModeliSGIX(GL_FRAGMENT_LIGHT_MODEL_LOCAL_VIEWER_SGIX,
0);
    glFragmentLightModelfSGIX(GL_FRAGMENT_LIGHT_MODEL_TWO_SIDE_SGIX, 0);
#endif
#endif
```

```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glRasterPos2i (0,0);

    glLightModelfv (GL_LIGHT_MODEL_AMBIENT, def_amb);

#ifdef GL_SGIX_fragment_lighting
#ifdef NEW_FRAG_SPEC
    glFragmentLightModelfvSGIX (GL_FRAGMENT_LIGHT_MODEL_AMBIENT_SGIX,
def_amb);
#endif

  glDisable(GL_FRAGMENT_LIGHTING_SGIX);
  glDisable(GL_FRAGMENT_LIGHT0_SGIX);
#endif

  glDisable(GL_LIGHTING);
  glDisable(GL_LIGHT0);
  glDisable (GL_DEPTH_TEST);
  glDisable (GL_NORMALIZE);
  glDisable (GL_BLEND);
#endif
}

@


1.4
log
@no new constants
@
text
@d1 2
d12 1
d15 3
a17 2
#include "tgeom.h"
#include "handy.h"
d19 1
a19 1

d21 4
a24 1
"assert %s failed in line %d of file %s\n", #a, __LINE__, __FILE__);
d41 2
```

```
a49 2
double res = 15;

a105 58
void init_lights(void)
{
  float dif[4] = {.7,.0,.0,1.};
  float whi[4] = {1.,1.,1.,1.};
  float bla[4] = {0.,0.,0.,0.};
  float pos[4] = {0.,1.,1.,0.};

  glFragmentMaterialfvSGIX(GL_FRONT_AND_BACK, GL_AMBIENT, bla);
  glFragmentMaterialfvSGIX(GL_FRONT_AND_BACK, GL_DIFFUSE, dif);
  glFragmentMaterialfvSGIX(GL_FRONT_AND_BACK, GL_SPECULAR, bla);
  glFragmentMaterialfSGIX(GL_FRONT_AND_BACK, GL_SHININESS, 30.);

  glFragmentMaterialiSGIX(GL_FRONT_AND_BACK, GL_ENV_MAP_SGIX, GL_NONE);

  glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_AMBIENT, whi);
  glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_DIFFUSE, whi);
  glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_SPECULAR, whi);
  glFragmentLightfvSGIX(GL_FRAGMENT_LIGHT0_SGIX, GL_POSITION, pos);

  testassert ( GL_NO_ERROR == (glGetError()) );
  glLightEnviSGIX(GL_LIGHT_ENV_MODE_SGIX, GL_MODULATE);
  glEnable(GL_FRAGMENT_LIGHTING_SGIX);
  glEnable(GL_FRAGMENT_LIGHT0_SGIX);

  /* normal lights */

  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, bla);
  glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, whi);
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, whi);
  glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 30.);

  testassert ( GL_NO_ERROR == (glGetError()) );
  glLightfv(GL_LIGHT0, GL_AMBIENT, bla);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, bla);
  glLightfv(GL_LIGHT0, GL_SPECULAR, bla);
  glLightfv(GL_LIGHT0, GL_POSITION, pos);

  testassert ( GL_NO_ERROR == (glGetError()) );
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective (70., 1., 1., 100.);
```

```
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();

  glEnable (GL_DEPTH_TEST);
  glEnable (GL_NORMALIZE);

  glClearColor(0.0,0.0,0.0,0.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glLightModeli (GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
  glFragmentLightModeliSGIX (GL_FRAGMENT_LIGHT_MODEL_LOCAL_VIEWER_SGIX,
GL_TRUE);
}

d110 1
d119 1
a119 1
  GLint smagint[4] = {INT_MAX, INT_MAX, INT_MIN, INT_MAX};
d126 1
a126 1
  GLint   ipix[4];
d132 1
d136 5
d262 1
d277 1
d279 1
d368 2
d570 1
a570 1
  glGetIntegerv (GL_LIGHT_MODEL_AMBIENT, ipix);
d576 3
a578 1

d581 1
d584 3
d590 9
a603 7
     /* change tessalation */
    case XGKEY:
      res *= 2.0;
      break;
    case XLKEY:
      res /= 2.0;
      break;
d611 1
d614 1
d620 1
a620 4
```

```
  glGetFloatv (GL_MIN_RED_SGIX, &minval);
  glGetFloatv (GL_MAX_RED_SGIX, &maxval);

d628 1
d690 103
@


1.3
log
@tests gets and checks inf/nan stuff correctly
@
text
@d154 1
a154 1
  glFragmentLightModeliSGIX (GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
d269 7
a280 4
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, bla);
  glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, smag); /* need to use diffuse
because the alpha value of the final color is taken from diffuse alpha and we want to check
clamping of all r, g, b, a*/
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, smag);
  glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 30.);
a281 7
  /* CHECK GET */
  glGetMaterialfv (GL_FRONT, GL_DIFFUSE, pix);
  testassert ( (pix[0] == smag[0]) &&
               (pix[1] == smag[1]) &&
               (pix[2] == smag[2]) &&
               (pix[3] == smag[3]) );

d297 8
d309 1
a309 1
  /* ...check the pass-through of per-vertex result into frag */
d317 4
a320 5
  /* ..scene ambient was irrelevant in per-vertex because the ambient
     material was black; it'll get turned on
     in per-fragment b/c we're matching ambient color to per-vertex results.
     So we explicitly turn it off here. */
  glFragmentLightModelfvSGIX (GL_LIGHT_MODEL_AMBIENT, bla);
d333 3
a335 3
  /* ...should be same as per-vertex result (since we're just passing it thru) except
     that alpha now matches the per-fragment diffuse material alpha, which is in
     opaquebla */
a354 7
```

```
 /* CHECK GETS */
 glGetFragmentMaterialfvSGIX (GL_FRONT, GL_DIFFUSE, pix);
 testassert ( (pix[0] == smag[0]) &&
              (pix[1] == smag[1]) &&
              (pix[2] == smag[2]) &&
              (pix[3] == smag[3]) );

d387 1
d399 8
d608 1
a608 1
 /* this is not being handled correctly */
a619 2


@


1.2
log
@fixed NaN comparison problem
@
text
@d1 1
d39 2
d50 3
a52 1

d54 1
d80 10
a89 10
 {INF, NAN, INF, ZERO},
 {NAN, INF, NEGINF, ZERO},
 {INF, INF, NAN, ZERO},
 {INF, INF, INF, ZERO},
 {NAN, NEGINF, NEGINF, ZERO},
 {NEGINF, NAN, INF, ZERO},
 {NEGINF, NEGINF, NAN, ZERO},
 {NEGINF, NEGINF, NEGINF, ZERO},
 {INF, NEGINF, NAN, NAN},
 {NEGINF, INF, NAN, NAN},
d91 3
a93 3
 {NORM, NORM, ZERO, NAN},
 {INF, NEGINF, INF, 1.0/NORM},
 {NEGINF, INF, NEGINF, 1.0/NORM},
d113 4
a116 4
 glFragmentLightfvSGIX(GL_LIGHT0, GL_AMBIENT, whi);
```

```
  glFragmentLightfvSGIX(GL_LIGHT0, GL_DIFFUSE, whi);
  glFragmentLightfvSGIX(GL_LIGHT0, GL_SPECULAR, whi);
  glFragmentLightfvSGIX(GL_LIGHT0, GL_POSITION, pos);
d169 1
d176 1
d185 10
a194 1
  glClearColor(0.0,0.0,0.0,0.0);
d198 1
d279 7
d322 3
a324 3
  glFragmentLightfvSGIX(GL_LIGHT0, GL_AMBIENT, whi);
  glFragmentLightfvSGIX(GL_LIGHT0, GL_DIFFUSE, whi);
  glFragmentLightfvSGIX(GL_LIGHT0, GL_SPECULAR, whi);
d348 10
a357 3
  glFragmentLightfvSGIX(GL_LIGHT0, GL_AMBIENT, bla);
  glFragmentLightfvSGIX(GL_LIGHT0, GL_DIFFUSE, smag);
  glFragmentLightfvSGIX(GL_LIGHT0, GL_SPECULAR, smag);
d373 2
a374 2
  glFragmentLightfvSGIX(GL_LIGHT0, GL_DIFFUSE, bla);
  glFragmentLightfvSGIX(GL_LIGHT0, GL_SPECULAR, bla);
d594 16
d647 3
d700 6
@


.
1.1
log
@color range extension test - erp
@
text
@d32 1
d72 1
d241 3
a243 1
   the average*/
d370 5
a374 7
  /* todo: spotlight?*/

  glDisable (GL_FRAGMENT_LIGHTING_SGIX);
  glDisable (GL_FRAGMENT_LIGHT0_SGIX);
  glDisable (GL_LIGHTING);
  glDisable (GL_LIGHT0);
```

```
d484 2
a485 1
    if ( pix[0] != blend_output[i][0])
a488 2
    else
        fprintf (stderr, "good result %d\n", i);
d501 1
a501 1
    if ( pix[0] != blend_output[i][1])
d517 1
a517 1
    if ( pix[0] != blend_output[i][2])
a521 9
    /* ...test 1/A or division? todo */


    /*
    if ( (A/B) != blend_output[i][3])
        {
          test4error ("%f / %f did not equal %f\n", A, B, blend_output[i][3]);
        }
        */
d651 9
@
```